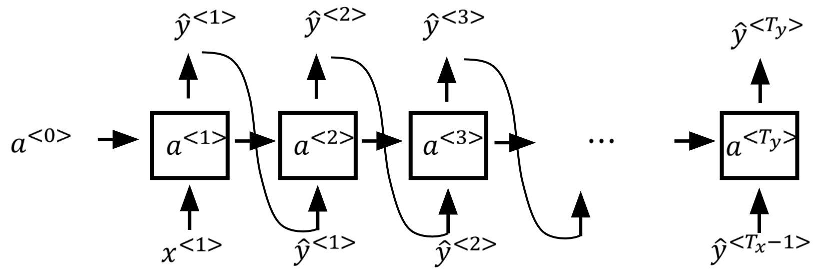
NLP

Week 5

Sequence to sequence model



$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$

Jane visite l'Afrique en septembre

$\longrightarrow$ Jane is visiting Africa in September.

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$

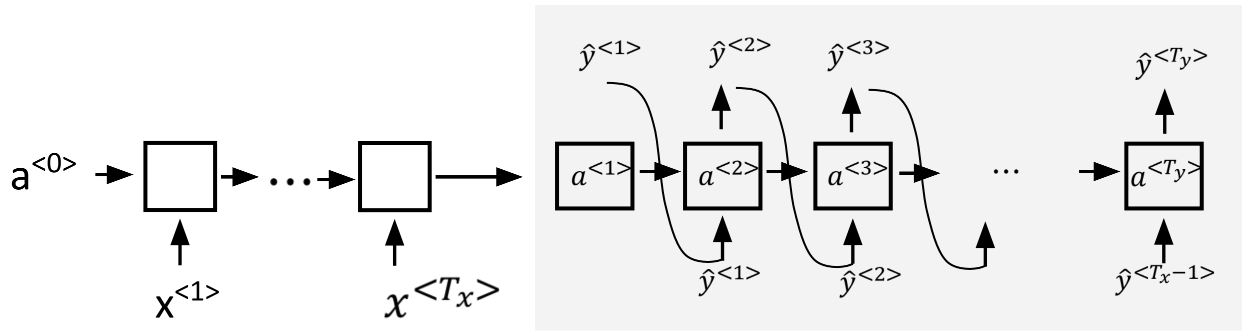
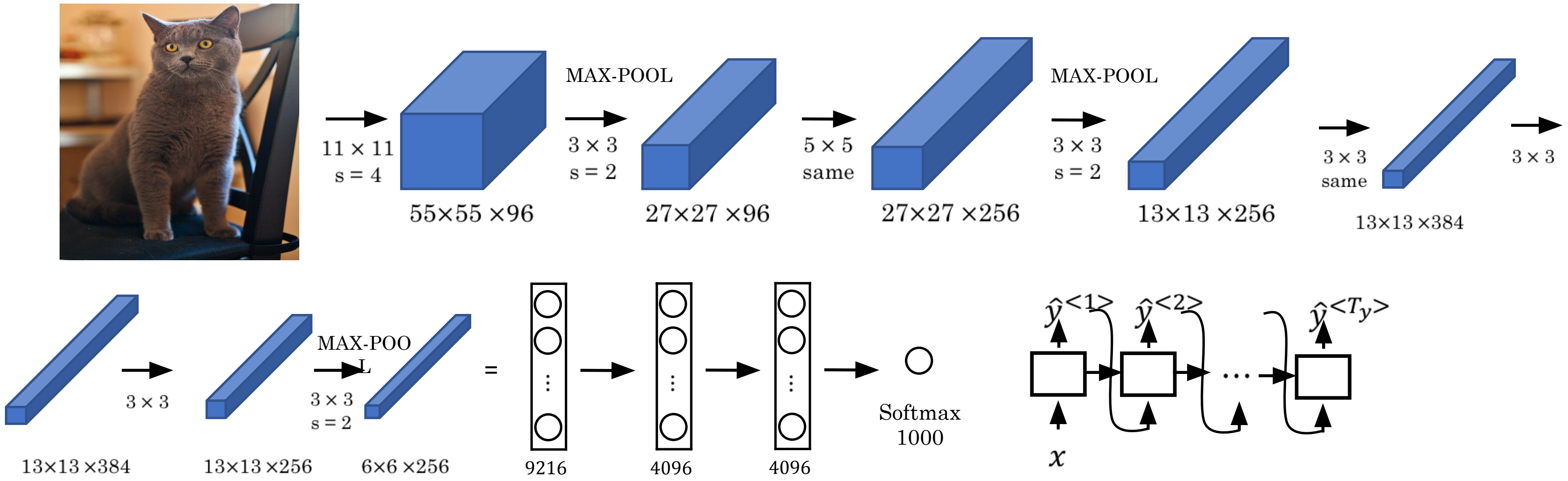


Image captioning

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$

A cat sitting on a chair



Finding the most likely translation

Jane visite l'Afrique en septembre.

$$P(y^{<1>}, \dots, y^{<T_y>} \mid x)$$

→ Jane is visiting Africa in September.

→ Jane is going to be visiting Africa in September.

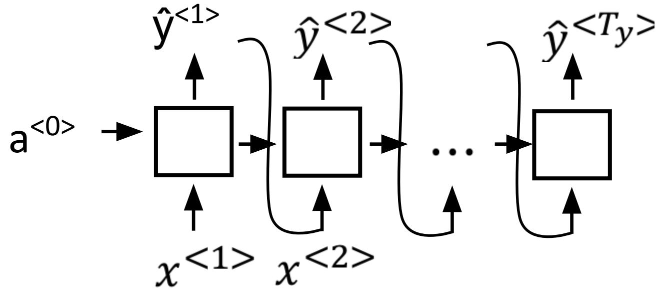
→ In September, Jane will visit Africa.

→ Her African friend welcomed Jane in September.

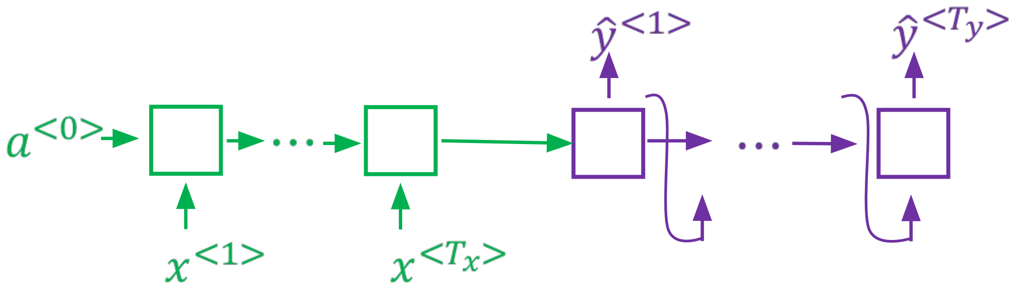
$$\underset{y^{<1>}, \dots, y^{<T_y>}}{\arg\max} P(y^{<1>}, \dots, y^{<T_y>} \mid x)$$

Machine translation as building a conditional language model

Language model:



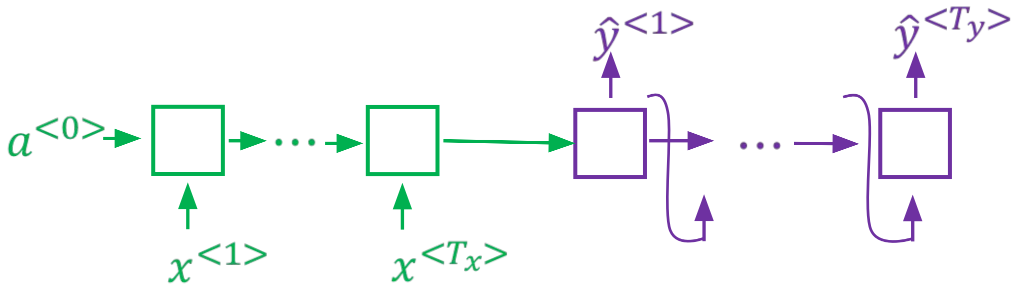
Machine translation:



Also called Conditional Language Model

$P(y^{<1>}, \ldots, y^{<T_y>} \mid x^{<1>}, \ldots, x^{<T_x>})$

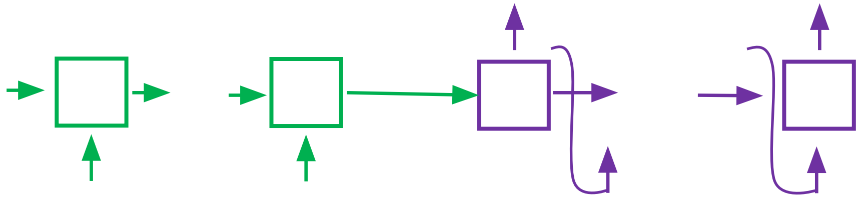
Why not a greedy search?



→ Jane is visiting Africa in September.

→ Jane is going to be visiting Africa in September.

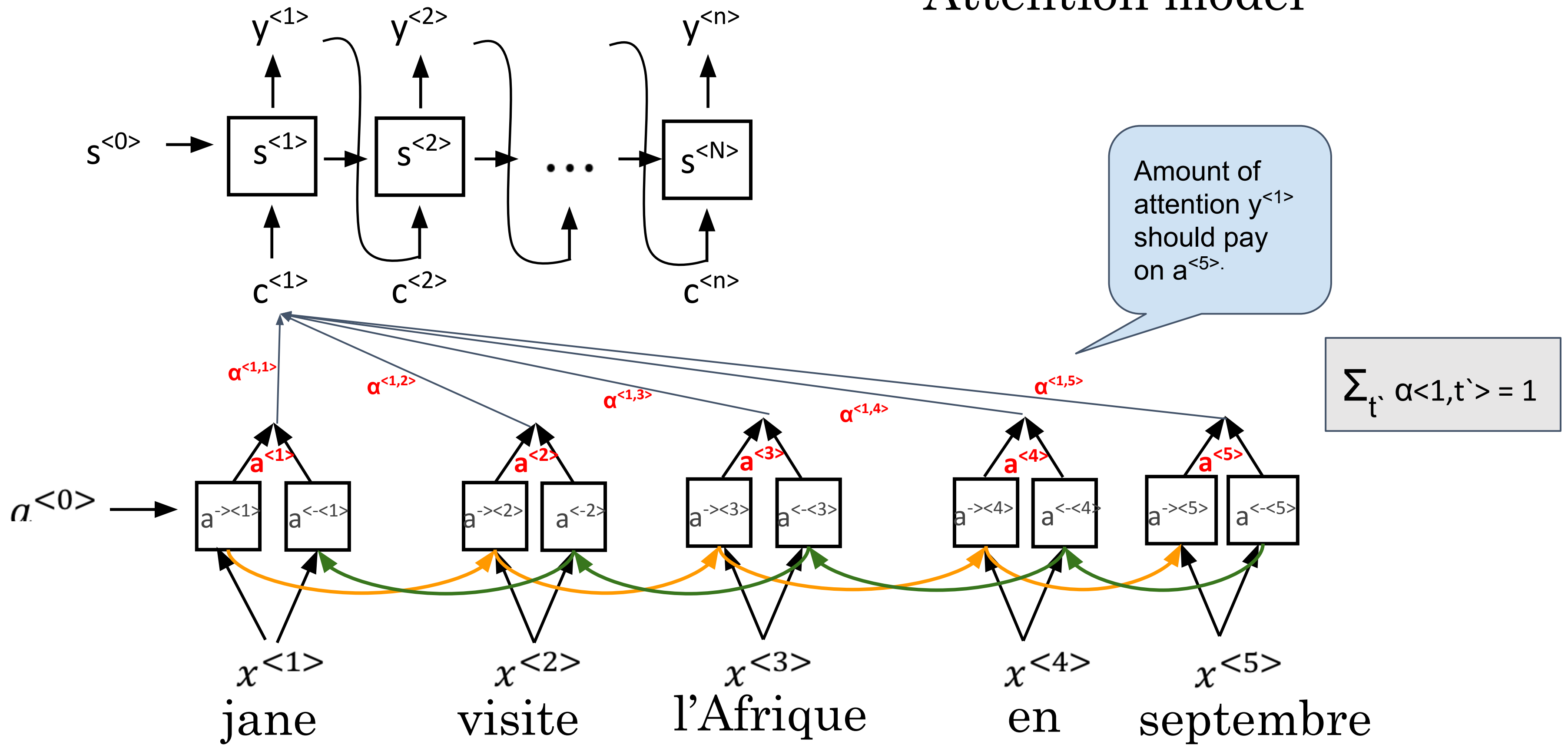
The problem of long sequences



Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.

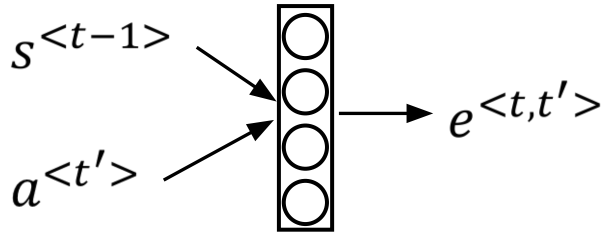
Attention model

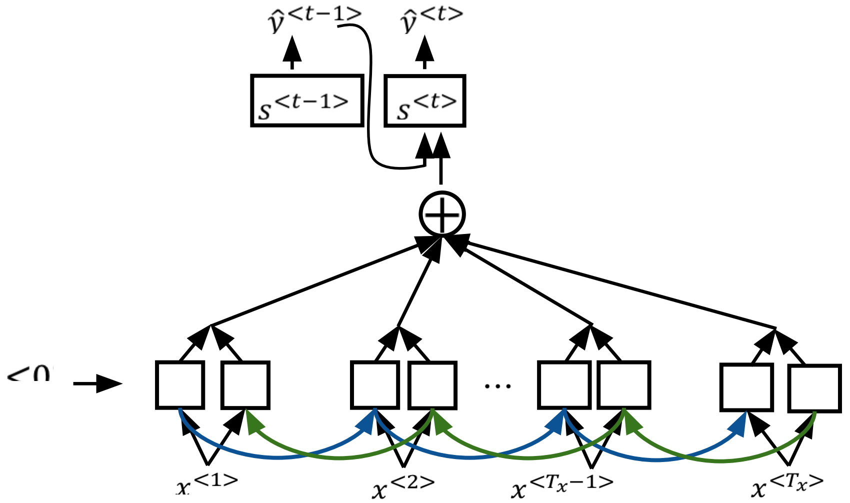


Computing attention $\alpha^{<t,t'>}$

$\alpha^{<t,t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$





Sequence to Sequence model

Transformers were developed to solve the problem of sequence transduction, or neural machine translation. That means any task that transforms an input sequence to an output sequence. This includes speech recognition, text-to-speech transformation, etc..

For models to perform sequence transduction, it is necessary to have some sort of memory.

“The Transformers” are a Japanese band. The band was formed in 1968, during the height of Japanese music history”

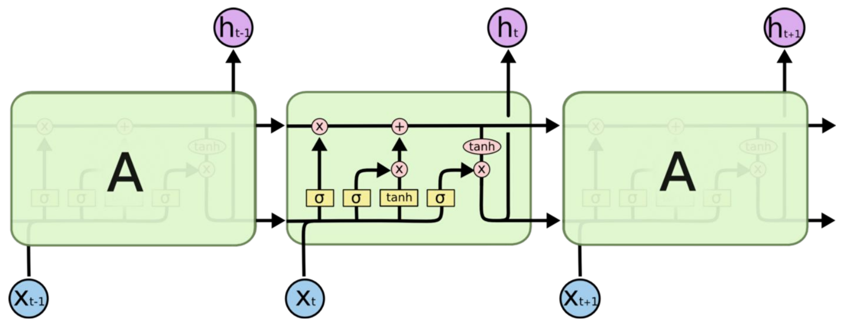
In this example, the word “the band” in the second sentence refers to the band “The Transformers” introduced in the first sentence. When you read about the band in the second sentence, you know that it is referencing to the “The Transformers” band. That may be important for translation. There are many examples, where words in some sentences refer to words in previous sentences.

SEQUENCE TO SEQUENCE MODEL

Sequence to Sequence model using Recurrent Neural Networks

Each word is processed separately, and the resulting sentence is generated by passing a hidden state to the decoding stage that, then, generates the output.

Long-Short Term Memory (LSTM)



When arranging one's calendar for the day, we prioritize our appointments. If there is anything important, we can cancel some of the meetings and accommodate what is important.

RNNs don't do that. Whenever it adds new information, it transforms existing information completely by applying a function. The entire information is modified, and there is no consideration of what is important and what is not.

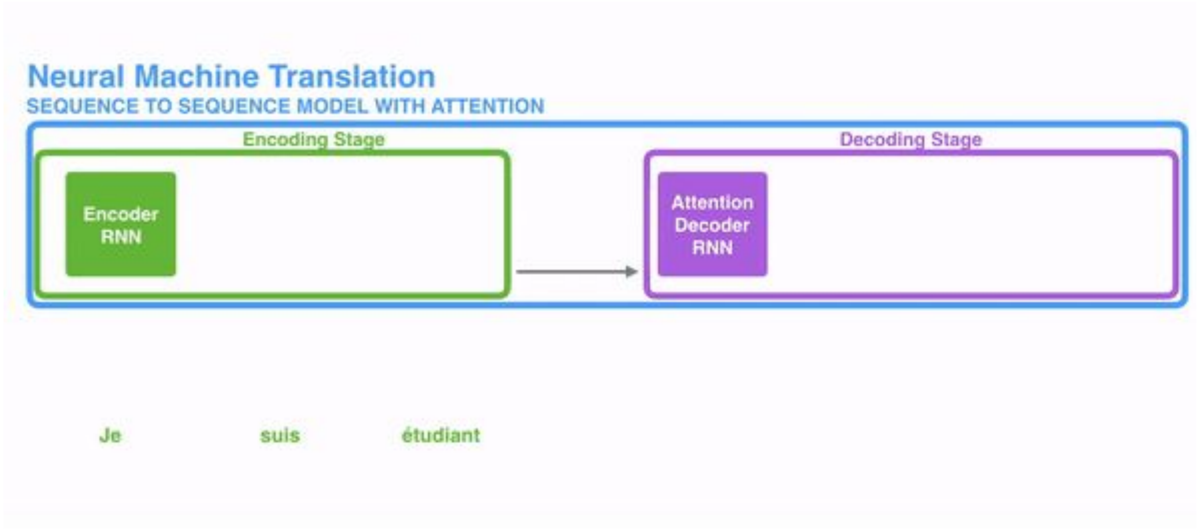
LSTMs make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. In this way, LSTMs can selectively remember or forget things that are important and not so important.

Sequential computation inhibits parallelization
No explicit modeling of long and short range dependencies
"Distance" between positions is linear

The reason for that is that the probability of keeping the context from a word that is far away from the current word being processed decreases exponentially with the distance from it.

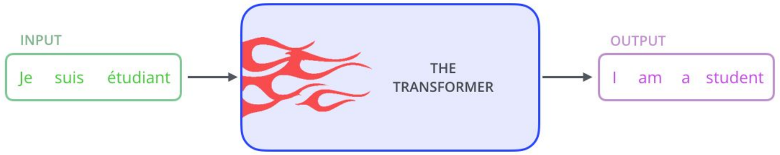
Attention

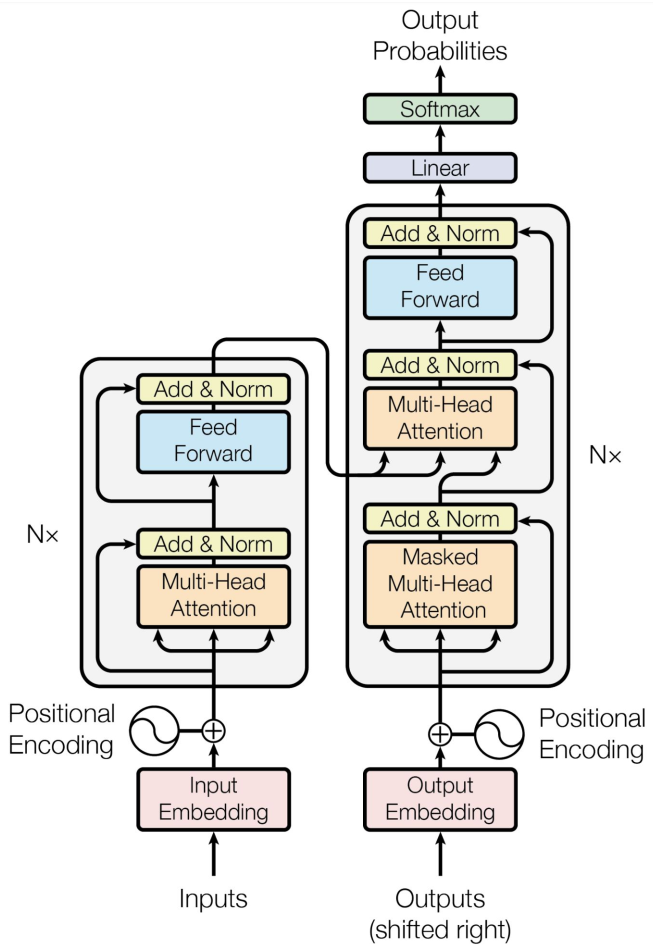
To solve these problems, Attention is a technique that is used in a neural network.
For RNNs, instead of only encoding the whole sentence in a hidden state, each word has a corresponding hidden state that is passed all the way to the decoding stage.



Transformer

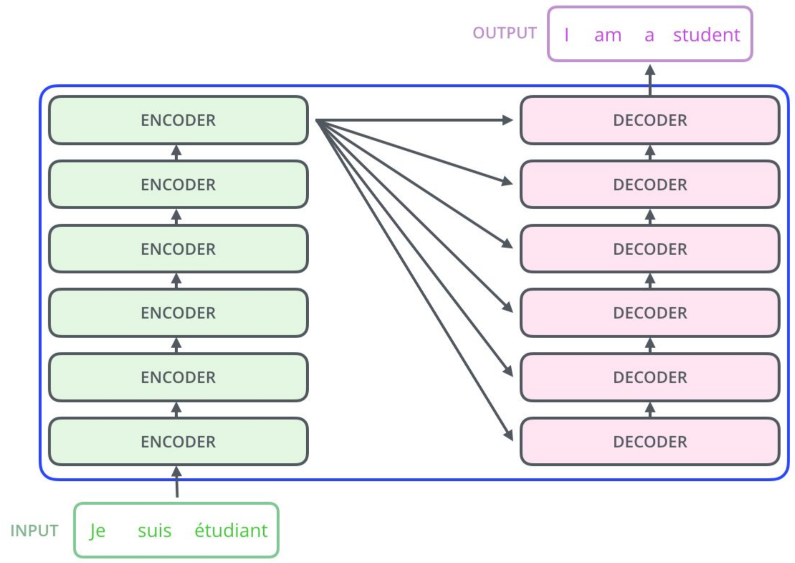
Transformer is a model that uses attention to boost the speed. More specifically, it uses self-attention.

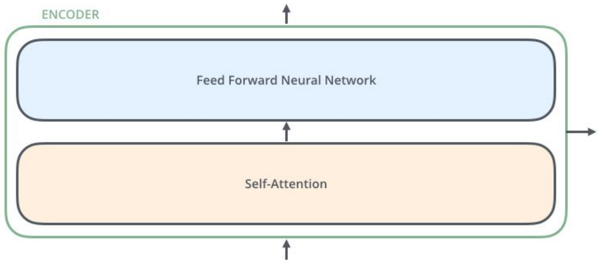




Transformer

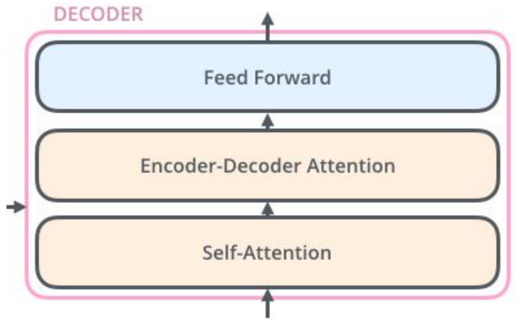
The Transformer consists of six encoders and six decoders.





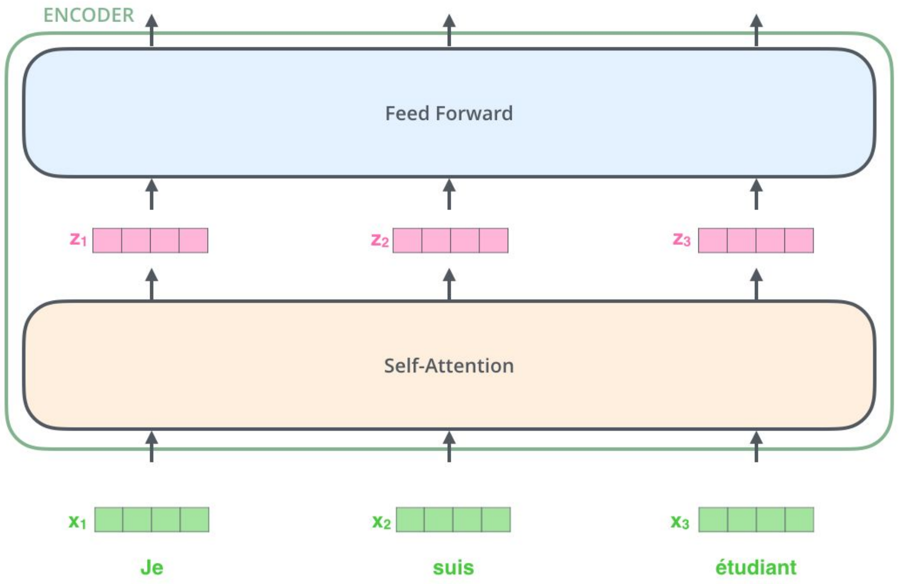
Each encoder consists of two layers: Self-attention and a feed Forward Neural Network.

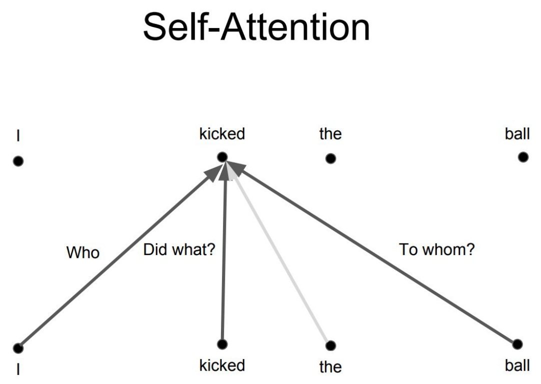
The encoder's inputs first flow through a self-attention layer. It helps the encoder look at other words in the input sentence as it encodes a specific word.



The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence.

Self-Attention

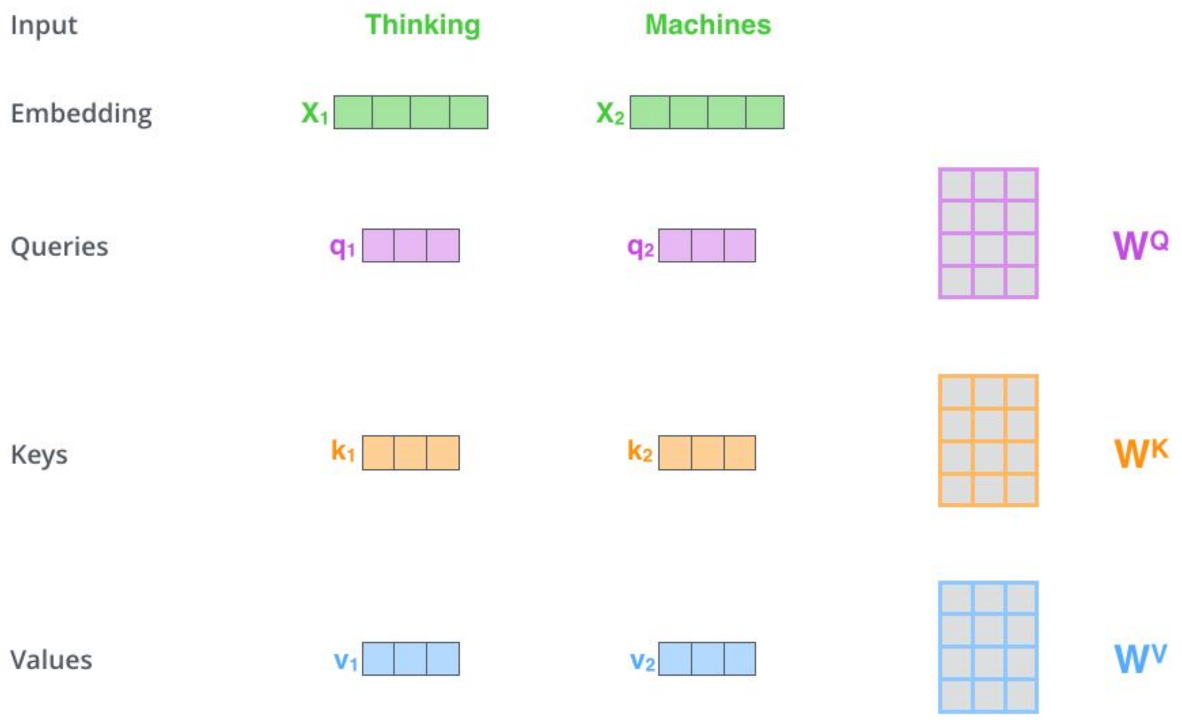




Each word is embedded into a vector of size 512. The embedding only happens in the bottom-most encoder.

There are dependencies between these paths in the self-attention layer. The feed-forward layer does not have those dependencies, however, and thus the various paths can be executed in parallel while flowing through the feed-forward layer.

Self-Attention - First Step



The first step in calculating self-attention is to create three vectors from each of the encoder's input vectors.

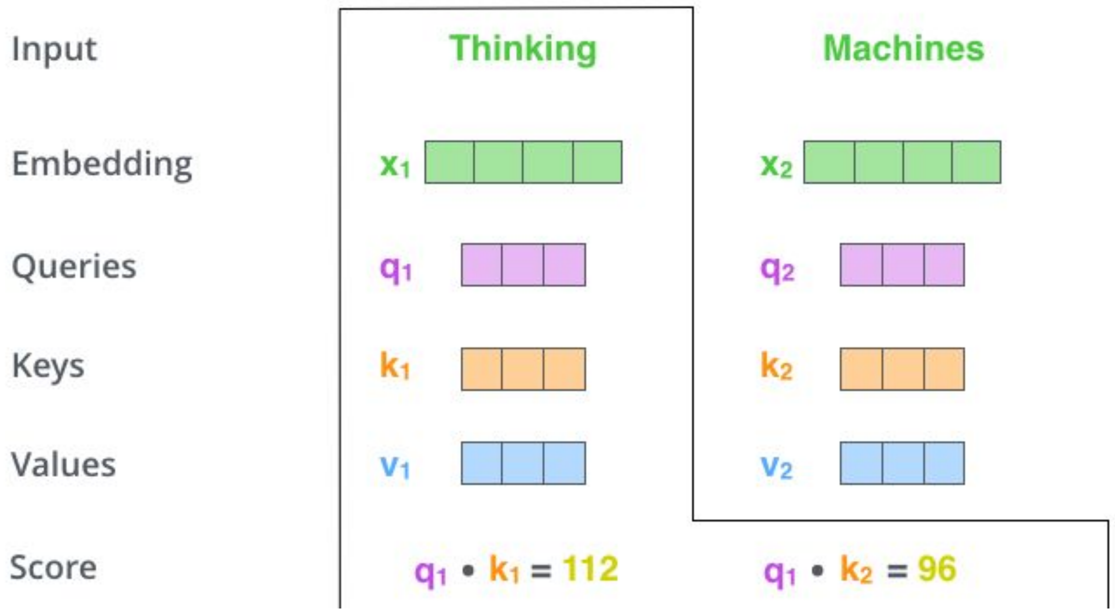
So for each word, we create a Query vector, a Key vector, and a Value vector. These vectors are created by multiplying the embedding by three matrices that we trained during the training process.

these new vectors are smaller in dimension than the embedding vector. Their dimensionality is 64, while the embedding and encoder input/output vectors have dimensionality of 512.

Multiplying x1 by the W^Q weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

What are the "query", "key", and "value" vectors? They're abstractions that are useful for calculating and thinking about attention.

Self-Attention - Second Step

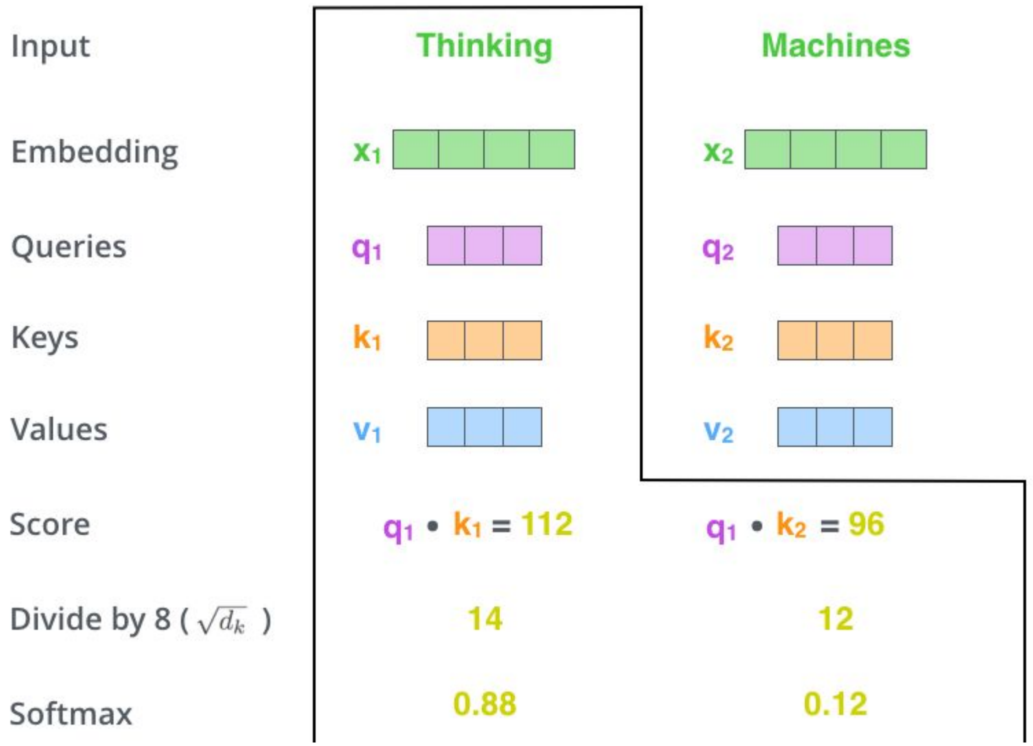


The second step in calculating self-attention is to calculate a score.

Say we're calculating the self-attention for the first word in this example, "Thinking". We need to score each word of the input sentence against this word. The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.

The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring. So if we're processing the self-attention for the word in position #1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.

Self-Attention - Third and Fourth steps

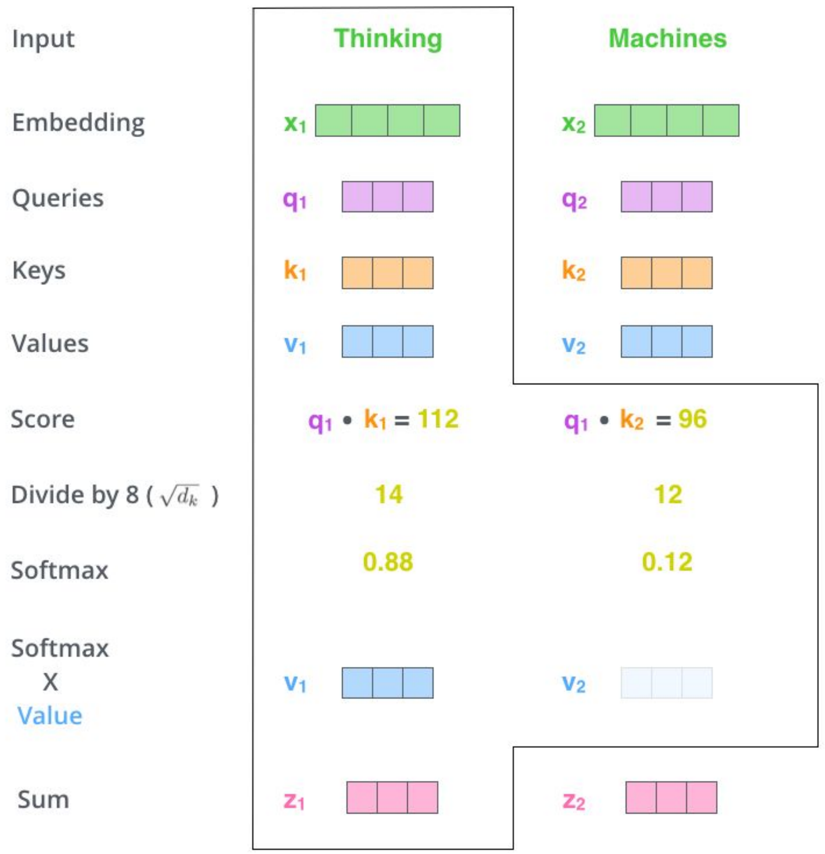


The third and fourth steps are to divide the scores by 8 (the square root of the dimension of the key vectors used in the paper — 64. This leads to having more stable gradients.

Then pass the result through a softmax operation. Softmax normalizes the scores so they're all positive and add up to 1.

This softmax score determines how much how much each word will be expressed at this position. Clearly the word at this position will have the highest softmax score, but sometimes it's useful to attend to another word that is relevant to the current word.

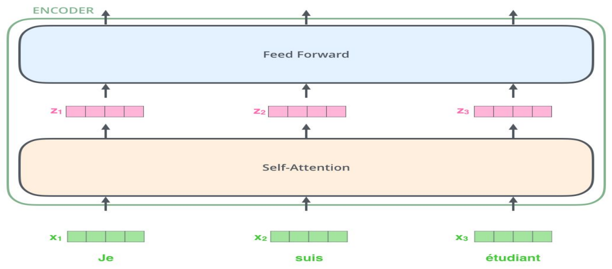
Self-Attention - Fifth and Sixth step



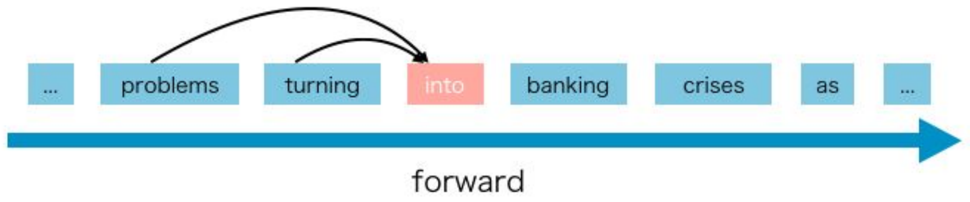
The fifth step is to multiply each value vector by the softmax score (in preparation to sum them up). The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words (by multiplying them by tiny numbers like 0.001, for example).

The sixth step is to sum up the weighted value vectors. This produces the output of the self-attention layer at this position (for the first word).

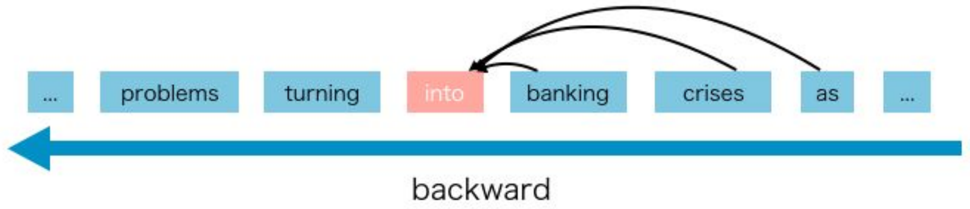
That concludes the self-attention calculation. The resulting vector is one we can send along to the feed-forward neural network. In the actual implementation, however, this calculation is done in matrix form for faster processing.



AR language model

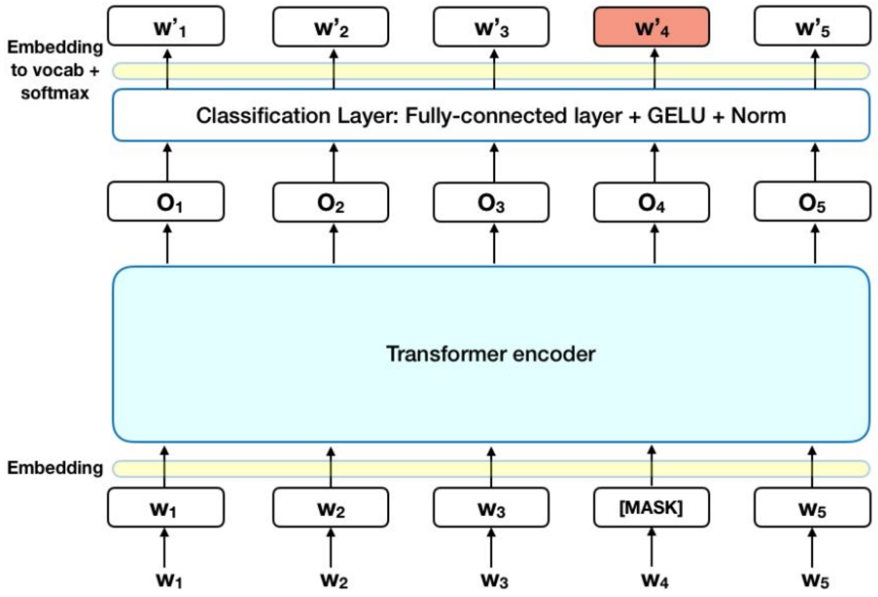


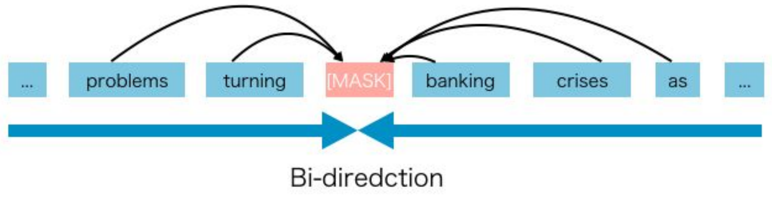
AR language model is a kind of model that using the context word to predict the next word. But here the context word is constrained to two directions, either forward or backward.



AR language model has some disadvantages, it only can use forward context or backward context, which means it can't use forward and backward context at the same time.

BERT


w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5



Unlike the AR language model, BERT is categorized as autoencoder(AE) language model.

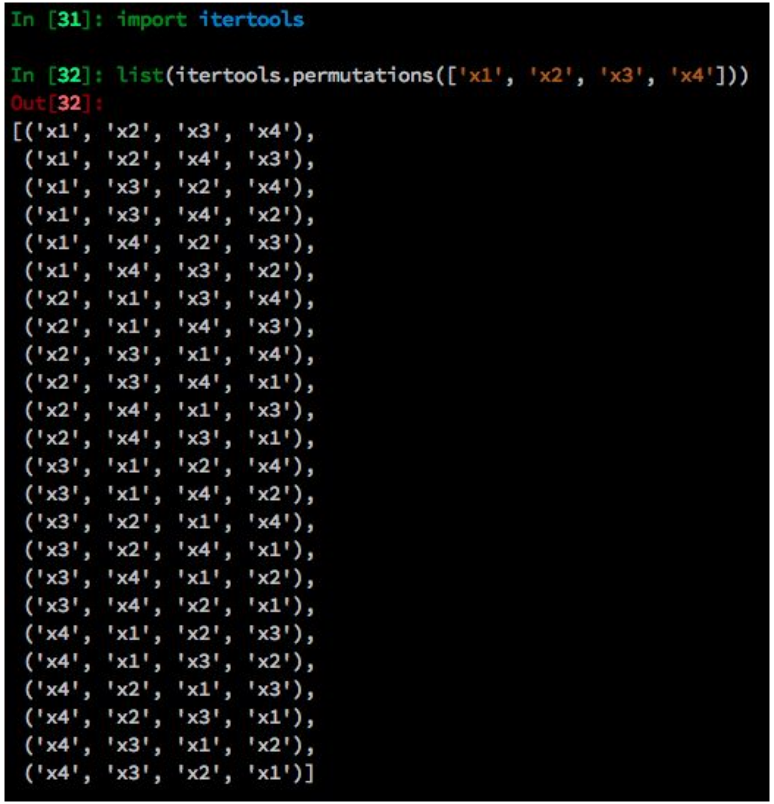
The AE language model aims to reconstruct the original data from corrupted input.

It uses the [MASK] in the pretraining, but this kind of artificial symbols are absent from the real data at finetuning time, resulting in a **pretrain-finetune discrepancy.**

Another disadvantage of [MASK] is that it assumes the predicted (masked) tokens are independent of each other given the unmasked tokens. EX:
It shows that the housing crisis was turned into a banking crisis

The advantages of AE language model is that it can see the context on both forward and backward direction.

XLNet



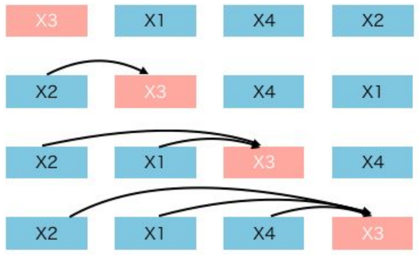
```
In [31]: import itertools

In [32]: list(itertools.permutations(['x1', 'x2', 'x3', 'x4']))
Out[32]:
[('x1', 'x2', 'x3', 'x4'),
 ('x1', 'x2', 'x4', 'x3'),
 ('x1', 'x3', 'x2', 'x4'),
 ('x1', 'x3', 'x4', 'x2'),
 ('x1', 'x4', 'x2', 'x3'),
 ('x1', 'x4', 'x3', 'x2'),
 ('x2', 'x1', 'x3', 'x4'),
 ('x2', 'x1', 'x4', 'x3'),
 ('x2', 'x3', 'x1', 'x4'),
 ('x2', 'x3', 'x4', 'x1'),
 ('x2', 'x4', 'x1', 'x3'),
 ('x2', 'x4', 'x3', 'x1'),
 ('x3', 'x1', 'x2', 'x4'),
 ('x3', 'x1', 'x4', 'x2'),
 ('x3', 'x2', 'x1', 'x4'),
 ('x3', 'x2', 'x4', 'x1'),
 ('x3', 'x4', 'x1', 'x2'),
 ('x3', 'x4', 'x2', 'x1'),
 ('x4', 'x1', 'x2', 'x3'),
 ('x4', 'x1', 'x3', 'x2'),
 ('x4', 'x2', 'x1', 'x3'),
 ('x4', 'x2', 'x3', 'x1'),
 ('x4', 'x3', 'x1', 'x2'),
 ('x4', 'x3', 'x2', 'x1')]
```

The XLNet propose a new way to let the AR language model learn from bi-directional context to avoid the disadvantages brought by the MASK method in AE language model.

A generalized autoregressive method that combines best of both of the AR and AE while avoiding their limitations.

Language model consists of two phases, the pre-train phase, and fine-tune phase. XLNet focus on pre-train phase. In the pre-train phase, it proposed a new objective called **Permutation Language Modeling**.



XLNet

The model will learn to gather information from all positions on both sides.

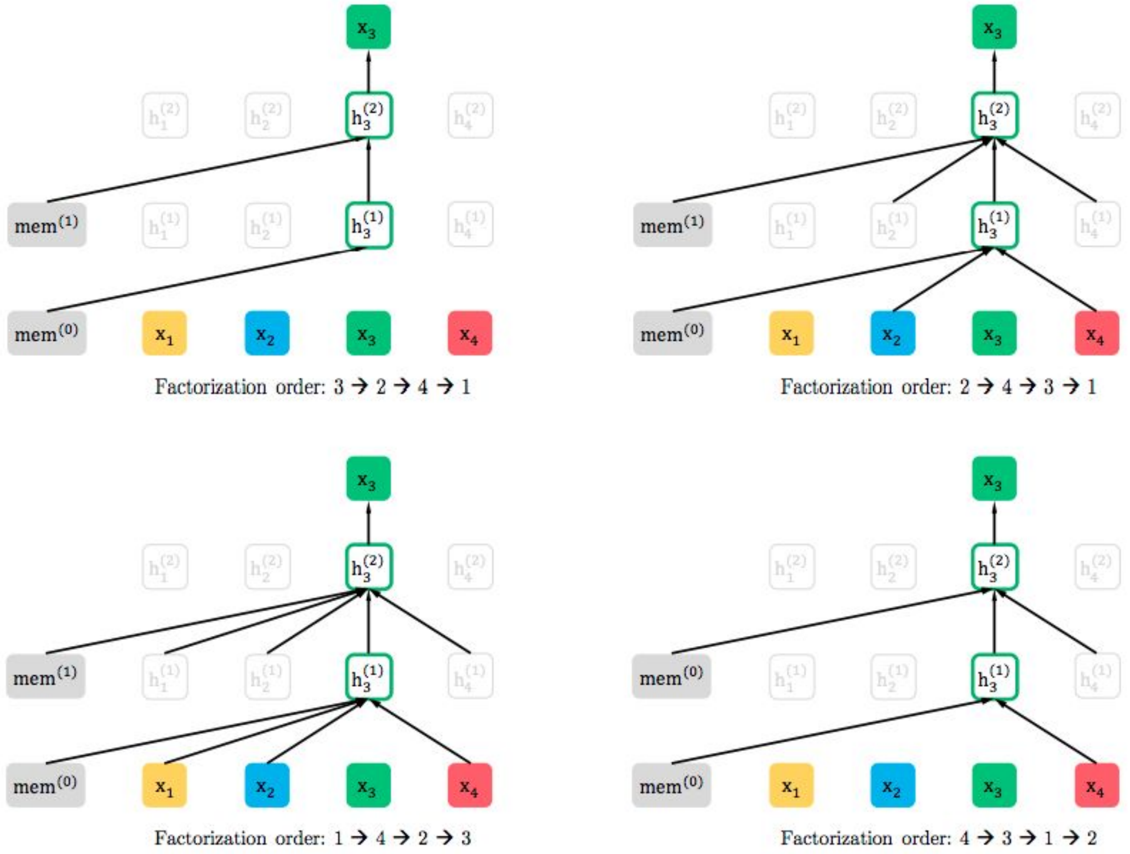
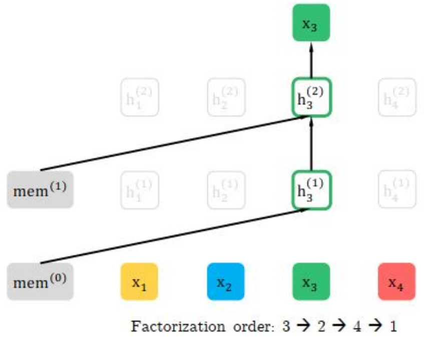

x_3
$h_1^{(2)}$
$h_2^{(2)}$
$h_3^{(2)}$
$h_4^{(2)}$
$mem^{(1)}$
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_4^{(1)}$
$mem^{(0)}$
x_1
x_2
x_3
x_4
Factorization order: 3 → 2 → 4 → 1
Factorization order: 2 → 4 → 3 → 1
Factorization order: 1 → 4 → 2 → 3
Factorization order: 4 → 3 → 1 → 2

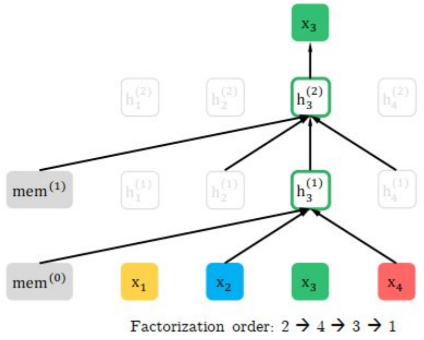
Figure 1: Illustration of the permutation language modeling objective for predicting x_3 given the same input sequence **x** but with different factorization orders.

Permutation Language Modeling - Objective function

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=1}^{T}\log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]$$



x_3
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$ $h_4^{(2)}$
mem$^{(1)}$
$h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
mem$^{(0)}$
x_1 x_2 x_3 x_4
Factorization order: 3 → 2 → 4 → 1

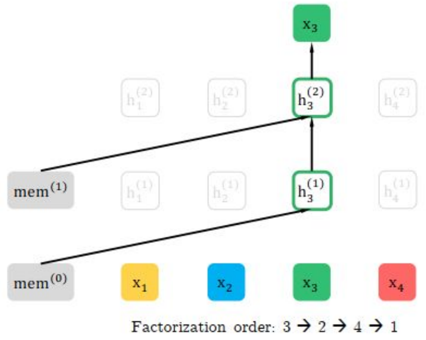


x_3
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$ $h_4^{(2)}$
mem$^{(1)}$
$h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
mem$^{(0)}$
x_1 x_2 x_3 x_4
Factorization order: 2 → 4 → 3 → 1

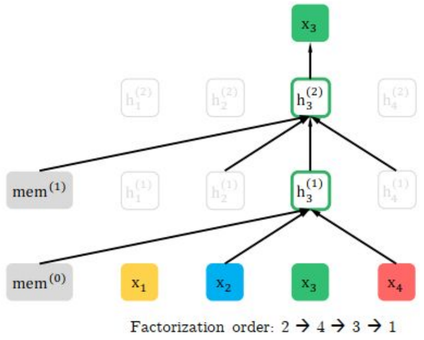
where Z is the set of all possible permutations of length t, z denotes t-th element and z<t denotes first t-1 elements of the permutation z ∈ Z

Permutation Language Modeling - Objective function

But, there is a problem. Suppose there are two factorization orders: z_1=1->**3**->4->2 and z_2=1->**3**->2->4. In both of these orders, first and second positions have the same values. If we want to predict the word at third position, we would take the same context at position 1 and 2 in both the orders. This would produce the same distribution of words in both cases regardless of the target position. So, we take the **context** and the **target position** and re-parameterise the **next-token distribution** to be target position aware as:



x_3
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$ $h_4^{(2)}$
$mem^{(1)}$ $h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
$mem^{(0)}$ x_1 x_2 x_3 x_4
Factorization order: 3 → 2 → 4 → 1



x_3
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$ $h_4^{(2)}$
$mem^{(1)}$ $h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
$mem^{(0)}$ x_1 x_2 x_3 x_4
Factorization order: 2 → 4 → 3 → 1

$$p_\theta(X_{z_t} = x \mid \mathbf{x}_{z_{<t}}) = \frac{\exp\left(e(x)^\top g_\theta(\mathbf{x}_{\mathbf{z}_{<t}}, z_t)\right)}{\sum_{x'} \exp\left(e(x')^\top g_\theta(\mathbf{x}_{\mathbf{z}_{<t}}, z_t)\right)}$$

There are two contradictory requirements for g-

1. To predict token at position z , g should only use the position z and not the content at z. Otherwise, the objective becomes trivial.
2. To predict tokens at positions greater than z, g should also encode content at the current position to provide full contextual information.

Two-Stream Self-Attention

Uses two sets of hidden representations:
Content stream representation h- encodes context and content at the position z. Initialized for the first layer by the corresponding word embedding at z.
Query stream representation g- encodes context and **position z**, but not the content at z. Initialized for the first layer by a learnable vector w.

$$\begin{aligned}
g_{z_t}^{(m)} &\leftarrow \text{Attention}(\mathrm{Q} = g_{z_t}^{(m-1)}, \mathrm{KV} = \mathbf{h}_{\mathbf{z}_{<t}}^{(m-1)}; \theta), \quad &&(\text{query stream: use } z_t \text{ but cannot see } x_{z_t}) \\
h_{z_t}^{(m)} &\leftarrow \text{Attention}(\mathrm{Q} = h_{z_t}^{(m-1)}, \mathrm{KV} = \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}; \theta), \quad &&(\text{content stream: use both } z_t \text{ and } x_{z_t}).
\end{aligned}$$

Q, K and V denote query, key and value respectively. The update is like the self-attention in Transformers.